~~06-14-01~~ 0320 ~~[illegible]~~

T

OIPE JC42 JUN 1 3 2001 PATENT & TRADEMARK OFFICE

Attorney Docket No.: 013777.002

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: William George Krieski, et al.

Serial No.: 09/840542

Filed: April 23, 2001

Title: Protocol Monitor
Commissioner for Patents
U.S. Patent & Trademark Office
Washington, D.C. 20231

TRANSMITTAL OF FORMAL DRAWINGS

Please find attached:

(a) the formal drawings for this application
Number of Sheets 43

[illegible]
SIGNATURE OF ATTORNEY

Reg. No.: 44,985

Bentley J. Olive
Type or print name of attorney

Tel. No.: (919) 286-8000

2200 W. Main Street, Suite 800
Address

Durham, North Carolina 27705

09840542.061301

DUR1\289630_ 1

Docket No. 013777.002

CERTIFICATE OF MAILING (37 C.F.R. § 1.8(a))

Date of Deposit: June 13, 2001

Inventor: William George Krieski, et al.

Application No.: 09/840542

Filed: April 23, 2001

For: Protocol Monitor

I hereby certify that the following documents:

Certificate Of Mailing (37 C.F.R. § 1.8(a)), and

1. Transmittal of Formal Drawings;
2. Formal Drawings (43 pages); and
3. Receipt acknowledgment postcard

are being deposited with the United States Postal on the date indicated above with sufficient postage as express mail addressed to: Commissioner for Patents, U.S. Patent and Trademark Office, Washington, D.C. 20231.

Ben Bruer
(Name of Person Mailing Documents)

[illegible]
(Signature of Person Mailing Documents)

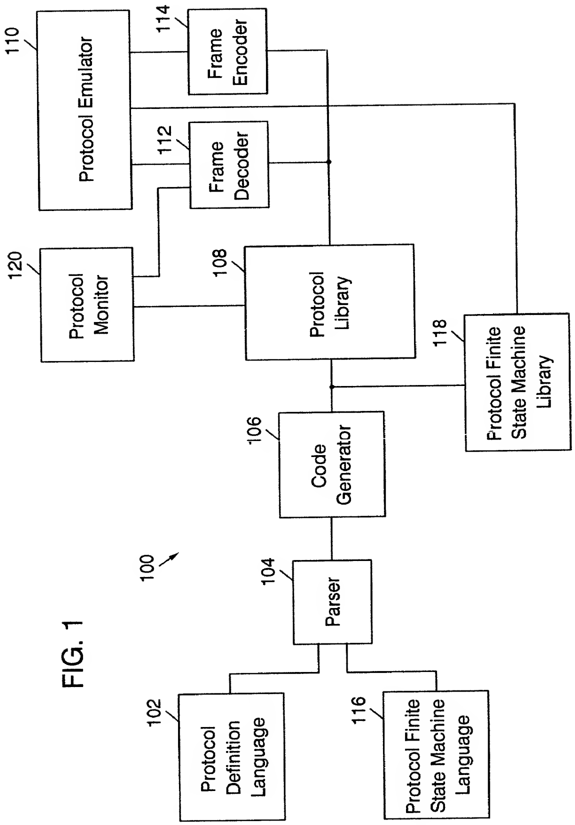
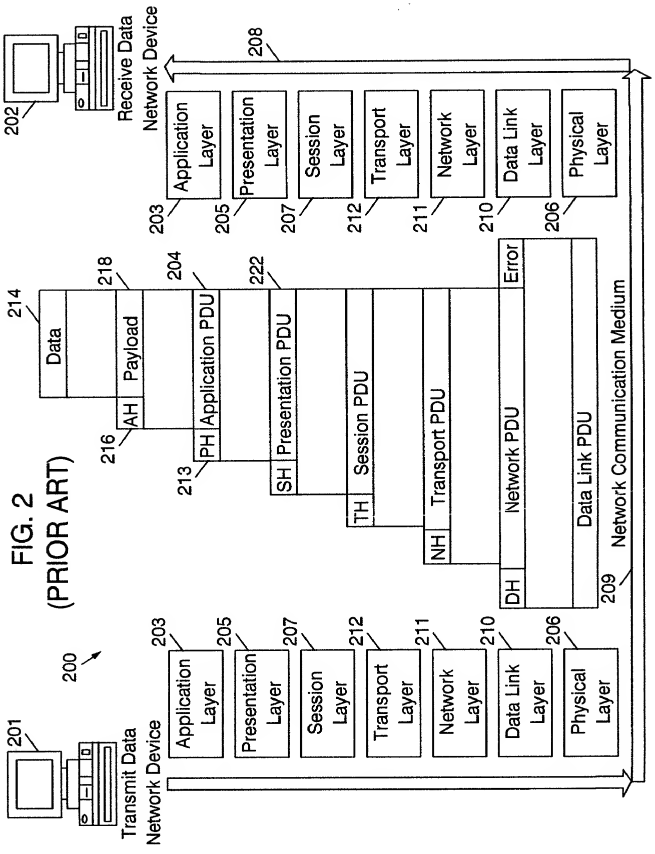

110
Protocol Emulator
114
Frame Encoder
112
Frame Decoder
120
Protocol Monitor
108
Protocol Library
118
Protocol Finite State Machine Library
106
Code Generator
100
104
Parser
102
Protocol Definition Language
116
Protocol Finite State Machine Language

FIG. 1

FIG. 2
(PRIOR ART)



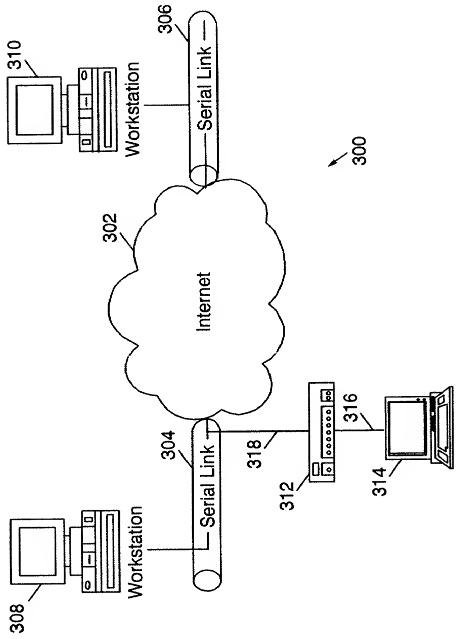
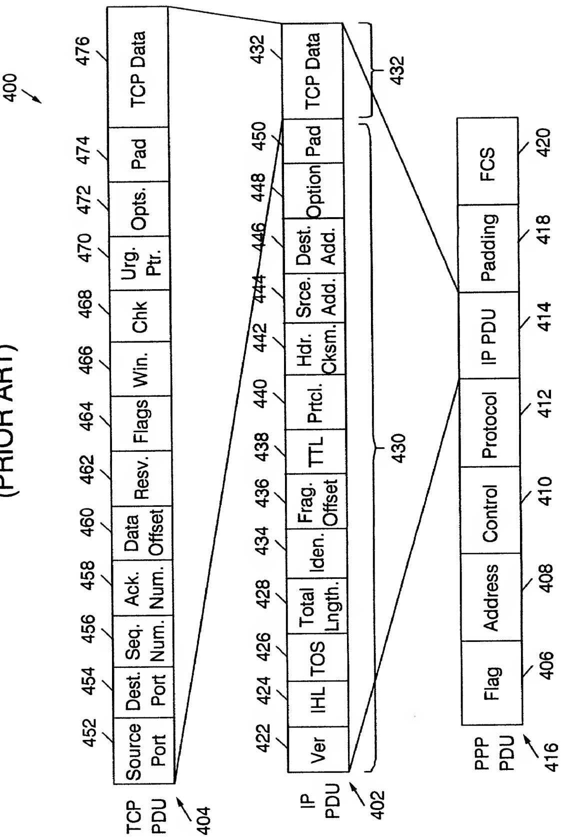


FIG. 3
(PRIOR ART)

FIG. 4
(PRIOR ART)



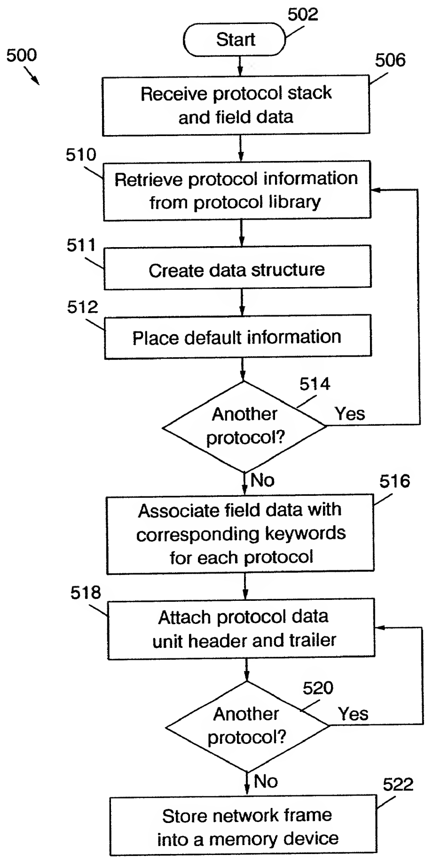

500
502
Start
506
Receive protocol stack and field data
510
Retrieve protocol information from protocol library
511
Create data structure
512
Place default information
514
Another protocol?
Yes
No
516
Associate field data with corresponding keywords for each protocol
518
Attach protocol data unit header and trailer
520
Another protocol?
Yes
No
522
Store network frame into a memory device

FIG. 5

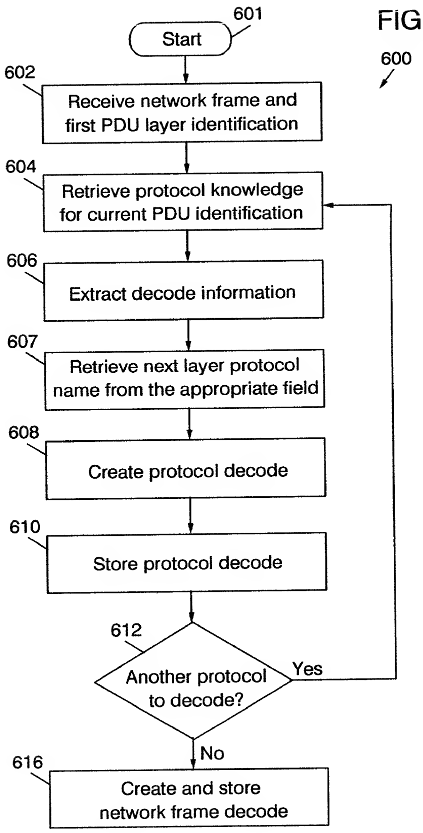

601
Start
602
Receive network frame and first PDU layer identification
604
Retrieve protocol knowledge for current PDU identification
606
Extract decode information
607
Retrieve next layer protocol name from the appropriate field
608
Create protocol decode
610
Store protocol decode
612
Another protocol to decode?
Yes
No
616
Create and store network frame decode
600

FIG. 6

FIG. 7

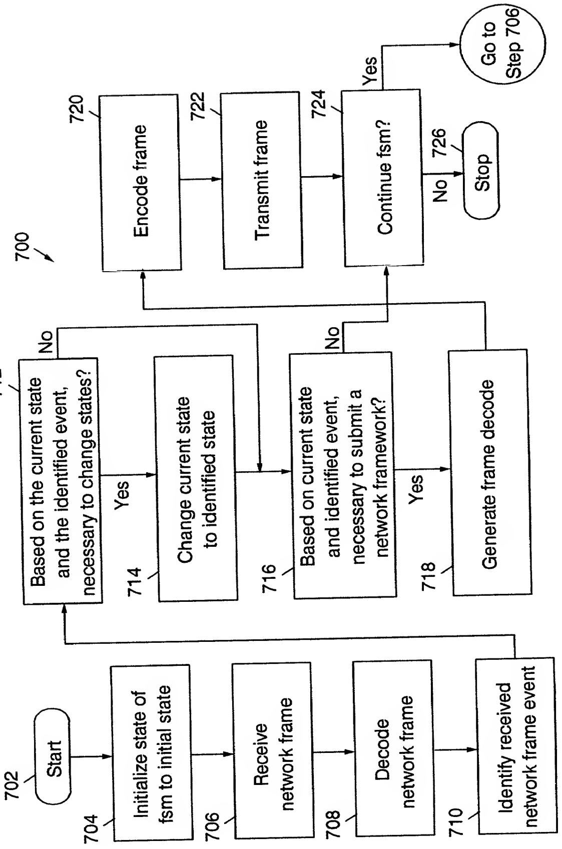

700
702
Start
704
Initialize state of fsm to initial state
706
Receive network frame
708
Decode network frame
710
Identify received network frame event
712
Based on the current state and the identified event, necessary to change states?
Yes
No
714
Change current state to identified state
716
Based on current state and identified event, necessary to submit a network framework?
Yes
No
718
Generate frame decode
720
Encode frame
722
Transmit frame
724
Continue fsm?
Yes
No
Go to Step 706
726
Stop

FIG. 8A

```
802
 protocol "IP" {// -------------------------
            len=valueof(field "Total Length")*8
            minLen=20*8 //just header
      804   maxLen=65535*8
          header "IP Header"
 806        payload "IP Payload"
      808
          header "IP Header" {// -----------------
810
            len=valueof(field "Header Length")*32
      812   field "Version"          818
      816   field "Header Length"
            compound_field "Type Of Service"
      814   field "Total Length"
      824                                           820
          field "Identification" {len=16 default=291}
          compound_field "Flags"                              822
 815      field "Fragment Offset" {len=13 desc="in 64 bits units"}   826
          field "Time To Live" {len=8 default=30 desc="seconds"}
          field "Protocol"            830
 828      field "Header Checksum"
          field "Source IP Address" {len=32 display=ipv4 field_type=
 832                must_encode}
          field "Destination IP Address" {
 834                    len=32
                        display=ipv4
                        field_type = must_encode
            }
```

FIG. 8B

816

```
repeat {
     len=valueof(field "Header Length") - 5 )*32//includes padding
   compound_field "Options"
  }

  field "Version" {
                  len=4
                  default=4
                  possible_values={
      0,15:"Reserved"
      1-3:"Unassigned"
                  6-14:"Unassigned"
  4:"IP Internet Protocol"
  5:"ST ST Datagram Mode"
  }}

  field "Header Length" {
                  len=4
                  minValue=5
                  desc="in 32 bit units"
                  default=eval_fn(len, "IP", "IP Header", "/32")
  }

  field "Total Length" {
                  minValue=20
                  len=16
                  desc="in octets include header length"
                  default=eval_fn(len, "IP", "IP", "/8")
  }
  field "Header Checksum" {
                  len=16
                  default=eval_fn(checksum, "IP", "IP Header")
                  display=hex
  }
```

FIG. 8C

```
compound_field "Type Of Service" { // ---------
                display=hex
                field "precedence" {
                len=3
                possible_values= {
0:"Routine"
1:"Priority"
2:"Immediate"
3:"Flash"
4:"Flash override"
5:"CRITIC/ECP"
6:"Internetwork Control"
7:"Network Control"
}}

field "Delay" {
len=1
                possible_values={0:"normal" 1:"low"}}

field "Throughput" {
                len=1
possible_values={0:"normal" 1:"high"}}

field "Reliability" {
                len=1
possible_values={0:"normal" 1:"high"}}

field "Monetary Cost" {
                len=1
possible_values={0:"normal" 1:"low"}}

field "Unused" {
                len=1
                possible_values={0:"valid"}}

}// end of field "Type of Service" -----------
```

FIG. 8D

```
compound_field "Flags" {
                len=3
                display=hex
field "Reserved" {
                len=1
                possible_values={0:"valid"}}
field "Fragment" {
                len=1
                possible_values={0:"May Fragment" 1:"Don't Fragment"}}
field "Fragments" {
                len=1
                possible_values={0:"last" 1:"more"}}
}
compound_field "Options" {//------------
     optional = (valueof(field "Header Length") > 5)
     compound_field "Option Tuple"
{
len=8;
display=hex
field "Copied Flag" {
                len=1
                possible_values={0:"not copied into all fragments
             0:"not copied into all fragments on fragmentation"
     1:"copied into all fragments on fragmentation"
}}
field "Option Class" {
                len=2
                possible_values={
                0:"control"
     1:"reserved for future use"
                2."debugging and measurement"
                3:"reserved for future use"
}}
```

FIG. 8E

```
field "Option Number" {
                len=5
                field_type=mulopt_other_fld
                possible_values={
                0:"end of option list"
          1:"no operation"
                2:"security"
                3:"loose source routing"
          4:"internet timestamp"
                7:"record route"
          8:"stream ID"
                9:"strict source routing"
}}
}

switch(valueof(field "Option Number")){
  0:null
  1:null
  2:compound_field "Security"
  3:compound_field "Loose Source Routing"
  9:compound_field "Strict Source Routing"
  7:compound_field "Record Route"
  8:compound_field "Stream ID"
  4:compound_field "Internet Timestamp"
}

compound_field "Security"{
                len=80
                field "Security Length" {
                        len=8
                        possible_values={0x0b:"valid"}}
```

FIG. 8F

```
                field "Security: Security"
                field "Compartments" {len=16}
                field "Handling Restrictions" {len=16}
                field "Transmission Control Code" {len=24}

                field "Security Security" {
                len=16
                possible_values={
                0:"unclassified"
                0xf135:"confidential"
                0x0789a:"EFTO"
                0xbc4d:"MMMM"
                0x5e26:"PROG"
                0xaf13:"Restricted"
                0xd788:"Secret"
                0x6bc5:"Top Secret"
              0x35e2,0x9af1,0x4d78,0x24bd,0x135e,0x89af,0xc4d6,0xe26b:
                "Reserved for future use"
   }}
  }

   compound_field "Strict Source Routing" {
    len=(valueof(field "Strict Source Routing Length")-1*8
    field "Strict Source Routing Length" {len=8 }
    field "Strict Source Routing Pointer" {len=8 minValue=4}

   repeat {
    len=(valueof(field "Strict Source Routing Length")-3)*8
    field "source address" {len=32 display=ipv4}
    }
  }
```

FIG. 8G

```
compound_field "Loose Source Routing" {
  len=(valueof(field "Loose Source Routing Length")-1*8
  field "Loose Source Routing Length" {len=8 }
  field "Loose Source Routing Pointer" {len=8 minValue=4}
 repeat {
  len=(valueof(field "Loose Source Routing Length")-3)*8
  field "source address" {len=32 display=ipv4}
  }
}
 compound_field "Record Routing" {
  len=(valueof(field "Record Routing Length")-1)*8
  field "Record Routing Length" {len=8 }
  field "Record Routing Pointer" {len=8 minValue=4}
 repeat {
  len=(valueof(field "Record Routing Length")-3)*8
  field "source address" {len=32 display=ipv4}
  }
}
 compound_field "Stream ID" {
  len=24
  field "Stream ID Length" {
     len=8
                 default=4
                 possible_values=
                        0x04:"valid"
            }}
  field "ID" {len=16 default=4}
}
```

FIG. 8H

```
      compound_field "Internet Timestamp" {
       field "Internet Timestamp Length" {len=8 }
       field "Internet Timestamp Pointer" {len=8 }
       field "Overflow" {
                 len=4
         desc="number of IP modules that cannot register timestamps"
                 }
       field "Flag" {
                 len=4
                 possible_values=1
         0:"time stamps only, stored in consecutive 32-bit words"
         1:"each timestamp is preceded with internet address"
         3:"the internet address fields are prespecified"
       }}

     } // end of Internet Timestamp
  } // end of field "option" ------------------

  } // end of field "IP" ----------------------

  field "Protocol" {

  len=8
  default=255
  field_type = mulopt_prtcl_fld
  display=hex
  possible_values={ //---------------------
       0:"HOPOPT (IPv6 Hop-by-Hop Option)"
       1:"ICMP (Internet Control Message)"
       2:"IGMP (Internet Group Management)"
       3:"GGP (Gateway-to-Gateway)"
```

FIG. 8I

4:"IP (IP in IP encapsulation)"
5:"ST (Stream)"
6:"TCP"
7:"CBT"
8:"EGP (Exterior Gateway Protocol)"
9:"IGP (any private interior gateway)"
10:"BBN-RCC-MON (BBN RCC Monitoring)"
11:"NVP-II (Network Voice Protocol)"
12:"PUP"
13:"ARGUS"
14:"EMCON"
15:"XNET (Cross Net Debugger)"
16:"CHAOS"
17:"UDP"
18:"MUX (Multiplexing)"
19:"DCN-MEAS (DCN Measurement Subsystems)"
20:"HMP (Host Monitoring)"
21:"PRM (Field Radio Measurement)"
22:"XNS-IDP (XEROX NS IDP)"
23:"TRUNK-1 (Trunk-1)"
24:"TRUNK-2 (Trunk-2)"
25:"LEAF-1 (Leaf-1)"
26:"LEAF-2 (Leaf-2)"
27:"RDP (Reliable Data Protocol)"
28:"IRTP (Internet Reliable Transaction)"
29:"ISO-TP4 (ISO Transport Protocol Class 4)"
30:"NETBLT (Bulk Data Transfer Protocol)"
31:"MFE-NSP (MFE Network Services Protocol)"
32:"MERIT-INP (MERIT Internodal Protocol)"
33:"SEP (Sequential Exchange Protocol)"
34:"3PC (Third Party Connect Protocol)"
35:"IDPR (Inter-Domain Policy Routing Protocol)"
36:"XTP (XTP)"

FIG. 8J

37:"DDP (Datagram Delivery Protocol)"
38:"IDPR-CMTP (IDPR Control Message Transport Protocol)"
39:"TP++ (TP++ Transport Protocol)"
40:"IL (IL Transport Protocol)"
41:"IPv6 (IPv6)"
42:"SDRP (Source Demand Routing Protocol)"
43:"IPv6-Route (Routing Header for IPv6)"
44:"IPv6-Frag (Fragment Header for IPv6)"
45:"IDRP (Inter-Domain Routing Protocol)"
46:"RSVP (Reservation Protocol)"
47:"GRE (General Routing Encapsulation)"
48:"MHRP (Mobile Host Routing Protocol)"
49:"BNA"
50:"ESP (Encap Security Payload for IPv6)"
51:"AH (Authentication Header for IPv6)"
52:"I-NLSP (Integrated Net Layer Security TUBA)"
53:"SWIPE (IP with Encryption)"
54:"NARP (NBMA Address Resolution Protocol)"
55:"MOBILE (IP Mobility)"
56:"TLSP (Transport Layer Security Protocol)"
57:"SKIP"
58:"IPv6-ICMP (ICMP for IPv6)"
59:"IPv6-NoNxt (No Next Header for IPv6)"
60:"IPv6-Opts (Destination Options for IPv6)"
61:"AHP (Any Host Internal Protocol)"
62:"CFTP (CFTP)"
63:"ALN (Any Local Network)"
64:"SAT-EXPAK (SATNET and Backroom EXPAK)"
65:"KRYPTOLAN (Kryptolan)"
66:"RVD (MIT Remote Virtual Disk Protocol)"
67:"IPPC (Internet Pluribus Field Core)"
68:"ADFS (Any Distributed File System)"
69:"SAT-MON (SATNET Monitoring)"
70:"VISA (VISA Protocol)"

FIG. 8K

71:"IPCV (Internet Field Core Utility)"
72:"CPNX (Computer Protocol Network Executive)"
73:"CPHB (Computer Protocol Heart Beat)"
74:"WSN (Wang Span Network)"
75:"PVP (Field Video Protocol)"
76:"BR-SAT-MON (Backroom SATNET Monitoring)"
77:"SUN-ND (SUN ND PROTOCOL-Temporary)"
78:"WB-MON (WIDEBAND Monitoring)"
79:"WB-EXPAK (WIDEBAND EXPAK)"
80:"ISO-IP (ISO Internet Protocol)"
81:"VMTP"
82:"SECURE-VMTP"
83:"VINES"
84:"TTP"
85:"NSFNET-IGP"
86:"DGP (Dissimilar Gateway Protocol)"
87:"TCF"
88:"EIGRP"
89:"OSPF"
90:"Sprite-RPC (Sprite RPC Protocol)"
91:"LARP (Locus Address Resolution Protocol)"
92:"MTP (Multicast Transport Protocol)"
93:"AX.25 (AX.25 Frames)"
94:"IPIP (IP-within-IP Encapsulation Protocol)"
95:"MICP (Mobile Internetworking Control Pro)"
96:"SCC-SP (Semaphore Communications Sec. Pro)"
97:"ETHERIP (Ethernet-within-IP Encapsulation)"
98:"ENCAP (Encapsulation Header)"
99:"APES (Any Private Encryption Scheme)"
100:"GMTP"
101:"IFMP (Ipsilon Flow Management Protocol)"
102:"PNNI (PNNI over IP)"
103:"PIM (Protocol Independent Multicast)"
104:"ARIS"

FIG. 8L

```
        105:"SCPS"
        106:"QNX"
        107:"A/N (Active Networks)"
        108:"IPPCP (IP Payload Compression Protocol)"
        109:"SNP (Sitara Networks Protocol)"
        110:"Compaq-Peer (Compaq Peer Protocol)"
        111:"IPX-in-IP"
        112:"VRRP (Virtual Router Redundancy Protocol)"
        113:"PGM (PGM Reliable Transport Protocol)"
        114:"AHOP (Any 0-hop protocol)"
        115-254:"Unassigned"
        255:"Reserved"
  }} // end of field "protocol" ·-----------------

        } // end of field "IP header" --------------

836
  payload "IP Payload" {// -----------------
          switch(valueof(field "Protocol")) {
838           1:protocol "ICMP"
        2:protocol "IGMP"
        6:protocol "TCP"
        17:protocol "UDP"
        46:protocol "RSVP"
        47:protocol "GRE"
        89.protocol "OSPF"
        }
  } // end of packet "IP payload" --------------
}
```

FIG. 9A

```
*/
/*******************************************************************************
Constants
******************************************************************** ************* /
//========  LCP Options   ================================
int  OPT_PASSIVE   = 1;    // Don't die if we don't get a response
int  OPT_RESTART = 2;     // Treat 2nd OPEN as DOWN, UP
int  OPT_SILENT = 4;      // Wait for peer to speak first

//========  LCP States    ================================
int  INITIAL_STATE = 0;
int  STARTING_STATE = 1;
int  CLOSED_STATE = 2;
int  STOPPED_STATE = 3;
int  CLOSING_STATE = 4;
int  STOPPING_STATE = 5;
int  REQ_SENT_STATE = 6;
int  ACK_RCVD_STATE = 7;
int  ACK_SENT_STATE = 8;
int  OPENED_STATE = 9;

//========  LCP Events    ================================
int  UP_EVENT = 0;
int  DOWN_EVENT = 1;
int  OPEN_EVENT = 2;
int  CLOSE_EVENT = 3;
int  TIMEOUT_POS_EVENT = 4;
```

FIG. 9B

```
int  TIMEOUT_NEG_EVENT = 5;
int  RCV_CFG_REQ_POS_EVENT = 6;
int  RCV_CFG_REQ_NEG_EVENT = 7;
int  RCV_CFG_ACK_EVENT = 8;
int  RCV_CFG_NACK_EVENT = 9;
int  RCV_TERM_REQ_EVENT = 10;
int  RCV_TERM_ACK_EVENT = 11;
int  RCV_UNKN_CODE_EVENT = 12;
int  RCV_CODE_REJECT_POS_EVENT = 13;
int  RCV_CODE_REJECT_NEG_EVENT = 14;
int  RCV_ECHO_REQ_REPLY_EVENT = 15;

//========  Transition Constants  ===========================
int  TRANSITION_CNST_FALSE = 0:
int  TRANSITION_CNST_TRUE = 1:

902~fsm    "LCP"
    {
904~state  INITIAL_STATE                          924
926 {                                             ↙
   ~UP_EVENT    -                            CLOSED_STATE
928~OPEN_EVENT InitialStOpenEvent            STARTING_STATE

    } // INITIAL
```

FIG. 9C

```
906
    state STARTING_STATE
    {
    UP_EVENT
    \
              switch (enabledSilent())
    \
                   TRANSITION_CNST_TRUE:    StareingStUpEvEnabledSilentTrue
    STOPPED_STATE
                   TRANSITION_CNST_FALSE:   StareingStUpEvEnabledSilentFalse
    REQ_SENT_STATE
              }
    \
    CLOSE_EVENT      -
    } / / STARTING
908
    state CLOSED_STATE
    {
    DOWN_EVENT                                                        INITIAL_STATE
    \
              switch (enabledSilent())
    \
        {
\
            TRANSITION_CNST_TRUE:        ClosedStOpenEvEnabledSilentTRUE
STOPPED_STATE    \
            TRANSITION_CNST_FALSE:       ClosedStOpenEvEnabledSilentFALSE
REQ_SENT_STATE   \
        }
\
RCV_CFG_REQ_POS_EVENT          ClosedStRcvCfgReqPosEv           CLOSED_STATE
RCV_CFG_REQ_NEG_EVENT          ClosedStRcvCfgReqNegEv           CLOSED_STATE
RCV_CFG_ACK_EVENT              ClosedStRcvCfgAckEv              CLOSED_STATE
RCV_CFG_NACK_EVENT             ClosedStRcvCfgNackEv             CLOSED_STATE
RCV_CODE_REJECT_POS_EVENT      RcvCodeRejectPosEv               CLOSED_STATE
RCV_CODE_REJECT_NEG_EVENT      ClosedStRcvCodeRejectNegEv       CLOSED_STATE
RCV_ECHO_REQ_REPLY_EVENT       RcvEchoReqReplyEv                CLOSED_STATE

} // CLOSED
910
state STOPPED_STATE
{
DOWN_EVENT                     StoppedStDownEv                  STARTING_STATE
OPEN_EVENT
\
            switch(enabledRestart ())
\
        {
\
            TRANSITION_CNST_TRUE:    StoppedStOpenEvEnabledRestartTRUE    STOPPED_STATE
\
```

FIG. 9D

FIG. 9E

```
        }

\
CLOSE_EVENT                   -                               CLOSED_STATE
RCV_CFG_REQ_POS_EVENT         StoppedStRcvCfgReqPosEv         ACK_SENT_STATE
RCV_CFG_REQ_NEG_EVENT         StoppedStRcvCfgReqNegEv         REQ_SENT_STATE
RCV_CFG_ACK_EVENT             StoppedStRcvCfgAckEv            STOPPED_STATE
RCV_CFG_NACK_EVENT            StoppedStRcvCfgNackEv           STOPPED_STATE
RCV_CODE_REJECT_POS_EVENT     RcvCodeRejectPosEv              STOPPED_STATE
RCV_CODE_REJECT_NEG_EVENT     StoppedStRcvCodeRejectNegEv     STOPPED_STATE
RCV_ECHO_REQ_REPLY_EVENT      RcvEchoReqReplyEv               STOPPED_STATE

} // STOPPED

912
state CLOSING_STATE
{
DOWN_EVENT                    ClosingStDownEv                 INITIAL_STATE
OPEN_EVENT                    ClosingStOpenEv                 STOPPING_STATE
TIMEOUT_POS_EVENT             ClosingStTimeoutPosEv           CLOSING_STATE
TIMEOUT_NEG_EVENT             ClosingStTimeNegEv              CLOSED_STATE
RCV_TERM_ACK_EVENT            ClosingStRcvTermAckEv           CLOSED_STATE
RCV_CODE_REJECT_POS_EVENT     RcvCodeRejectPosEv              CLOSING_STATE
RCV_CODE_REJECT_NEG_EVENT     RcvCodeRejectNegEv              CLOSED_STATE
RCV_ECHO_REQ_REPLY_EVENT      RcvEchoReqReplyEv               CLOSING_STATE
} // CLOSING
```

FIG. 9F

```
914
  state STOPPING_STATE
  {
  DOWN_EVENT                      StoppingStDownEv          STARTING_STATE
  CLOSE_EVENT                     -                         CLOSING_STATE
  TIMEOUT_POS_EVENT               StoppingStTimeoutPosEv    STOPPING_STATE
  TIMEOUT_NEG_EVENT               StoppingStTimeNegEv       STOPPED_STATE
  RCV_TERM_ACK_EVENT              StoppingStRcvTermAckEv    STOPPED_STATE
  RCV_CODE_REJECT_POS_EVENT       RcvCodeRejectPosEv        STOPPING_STATE
  RCV_CODE_REJECT_NEG_EVENT       RcvCodeRejectNegEv        STOPPED_STATE
  RCV_ECHO_REQ_REPLY_EVENT        RcvEchoReqReplyEv         STOPPING_STATE
  } // STOPPING
916
  state REQ_SENT_STATE
  {
  DOWN_EVENT                      ReqSentStDownEv           STARTING_STATE
  CLOSE_EVENT                     ReqSentStCloseEv          CLOSING_STATE
  TIMEOUT_POS_EVENT               ReqSentStTimeoutPosEv     REQ_SENT_STATE
  TIMEOUT_NEG_EVENT               ReqSentStTimeNegEv        STOPPED_STATE
  RCV_CFG_REQ_POS_EVENT           ReqSentStRcvCfgReqPosEv   ACK_SENT_STATE
  RCV_CFG_REQ_NEG_EVENT           ReqSentStRcvCfgReqNegEv   REQ_SENT_STATE
  RCV_CFG_ACK_EVENT               ReqSentStRcvCfgAckEv      ACK_RCVD_STATE
  RCV_CFG_NACK_EVENT              ReqSentStRcvCfgNackEv     REQ_SENT_STATE
  RCV_CODE_REJECT_POS_EVENT       RcvCodeRejectPosEv        REQ_SENT_STATE
  RCV_CODE_REJECT_NEG_EVENT       RcvCodeRejectNegEv        STOPPED_STATE
  RCV_ECHO_REQ_REPLY_EVENT        RcvEchoReqReplyEv         REQ_SENT_STATE
  } // REQ_SENT_STATE
```

FIG. 9G

```
918
 ~ state ACK_RCVD_STATE
   {
   DOWN_EVENT                     AckRcvdStDownEv             STARTING_STATE
   CLOSE_EVENT                    AckRcvdStCloseEv            CLOSING_STATE
   TIMEOUT_POS_EVENT              AckRcvdStTimeoutPosEv       REQ_SENT_STATE
   TIMEOUT_NEG_EVENT              AckRcvdStTimeNegEv          STOPPED_STATE
   RCV_CFG_REQ_POS_EVENT          AckRcvdStRcvCfgReqPosEv     OPENED_STATE
   RCV_CFG_REQ_NEG_EVENT          AckRcvdStRcvCfgReqNegEv     ACK_RCVD_STATE
   RCV_CFG_ACK_EVENT              AckRcvdStRcvCfgAckEv        REQ_SENT_STATE
   RCV_CFG_NACK_EVENT             AckRcvdStRcvCfgNackEv       REQ_SENT_STATE
   RCV_TERM_REQ_EVENT             AckRcvdStRcvTermReqEv       REQ_SENT_STATE
   RCV_TERM_ACK_EVENT             -                           REQ_SENT_STATE
   RCV_UNKN_CODE_EVENT            -                           ACK_RCVD_STATE
   RCV_CODE_REJECT_POS_EVENT      RcvCodeRejectPosEv          REQ_SENT_STATE
   RCV_CODE_REJECT_NEG_EVENT      RcvCodeRejectNegEv          STOPPED_STATE
   RCV_ECHO_REQ_REPLY_EVENT       RcvEchoReqReplyEv           ACK_RCVD_STATE

   } // ACK_RCVD_STATE

920
 ~ state ACK_SENT_STATE
   {
   DOWN_EVENT                     AckSentStDownEv             STARTING_STATE
   CLOSE_EVENT                    AckSentStCloseEv            CLOSING_STATE
   TIMEOUT_POS_EVENT              AckSentStTimeoutPosEv       ACK_SENT_STATE
   TIMEOUT_NEG_EVENT              AckSentStTimeNegEv          STOPPED_STATE
```

FIG. 9H

```
    RCV_CFG_REQ_POS_EVENT           AckSentStRcvCfgReqPosEv          ACK_SENT_STATE
    RCV_CFG_REQ_NEG_EVENT           AckSentStRcvCfgReqNegEv          REQ_SENT_STATE
    RCV_CFG_ACK_EVENT               AckSentStRcvCfgAckEv             OPENED_STATE
    RCV_CFG_NACK_EVENT              AckSentStRcvCfgNackEv            ACK_SENT_STATE
    RCV_TERM_REQ_EVENT              AckSentStRcvTermReqEv            REQ_SENT_STATE
    RCV_CODE_REJECT_POS_EVENT       RcvCodeRejectPosEv               ACK_SENT_STATE
    RCV_CODE_REJECT_NEG_EVENT       RcvCodeRejectNegEv               STOPPED_STATE
    RCV_ECHO_REQ_REPLY_EVENT        RcvEchoReqReplyEv                ACK_SENT_STATE

    } // ACK_SENT_STATE

922 state OPENED_STATE
    {
    DOWN_EVENT                      OpenedStDownEv                   STARTING_STATE
    OPEN_EVENT
    \
            switch(enabledRestart ())
    \
          {
    \
            TRANSITION_CNST_TRUE:  OpenedStOpenEvEnabledRestartTRUE  OPENED_STATE
    \
          }
    \
```

FIG. 9I

```
CLOSE_EVENT                    OpenedStCloseEv                CLOSING_STATE
RCV_CFG_REQ_POS_EVENT          OpenedStCfgReqPosEv            ACK_SENT_STATE
RCV_CFG_REQ_NEG_EVENT          OpenedStRcvCfgReqNegEv         REQ_SENT_STATE
RCV_CFG_ACK_EVENT              OpenedRcvCfgAckEv              REQ_SENT_STATE
RCV_CFG_NACK_EVENT             OpenedStRcvCfgNackEv           REQ_SENT_STATE
RCV_TERM_REQ_EVENT             OpenedStRcvTermReqEv           STOPPING_STATE
RCV_TERM_ACK_EVENT             OpenedStRcvTermAckEv           REQ_SENT_STATE
RCV_CODE_REJECT_POS_EVENT      RcvCodeRejectPosEv             OPENED_STATE
RCV_CODE_REJECT_NEG_EVENT      OpenedStRcvCodeRejectNegEv     STOPPING_STATE
RCV_ECHO_REQ_REPLY_EVENT       RcvEchoReqReplyEv              OPENED_STATE

} // OPENED_STATE

}
```

FIG. 10

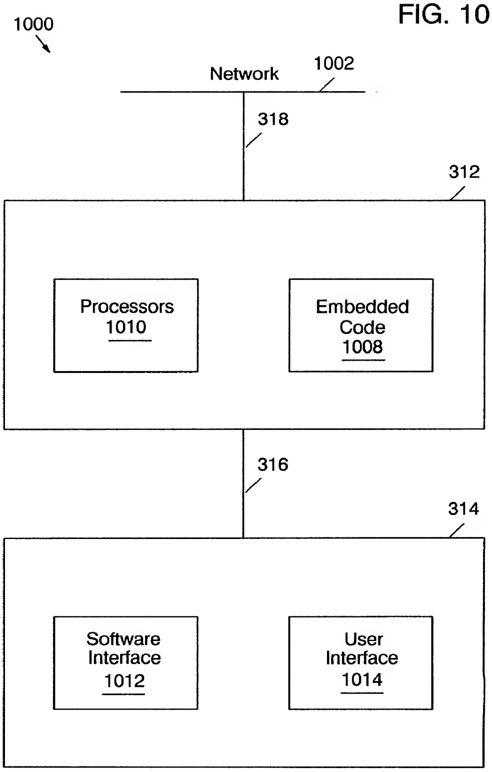

1000
Network
1002
318
312
Processors
1010
Embedded
Code
1008
316
314
Software
Interface
1012
User
Interface
1014

FIG. 11

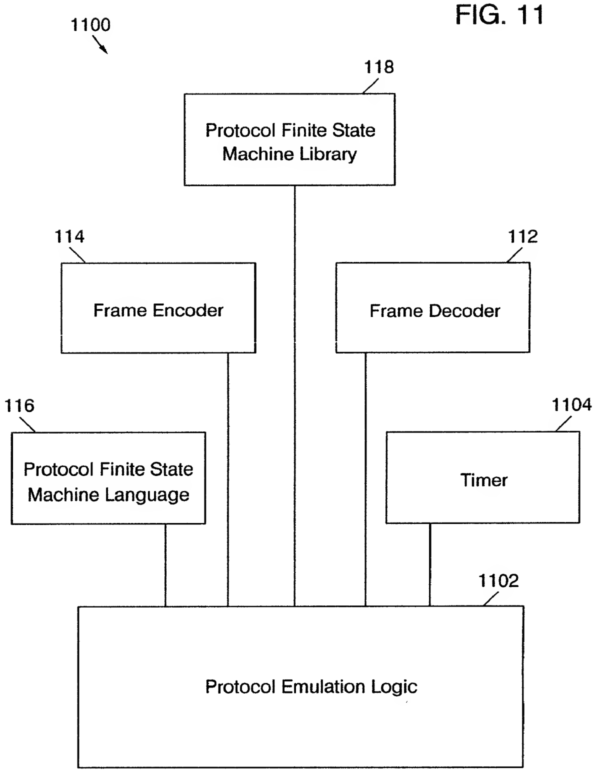

1100
118
Protocol Finite State Machine Library
114
Frame Encoder
112
Frame Decoder
116
Protocol Finite State Machine Language
1104
Timer
1102
Protocol Emulation Logic

FIG. 12A

1202

Events	State 0 Initial	1 Starting	2 Closed	3 Stopped	4 Closing	5 Stopping
Up	2	tc1,6	-	-	-	-
Down	-	-	0	1	0	1
Open	1	1	tc1,3/tc2,6	tc3,3r	5r	5r
Close	0	0	2	2	4	4
TO+	-	-	-	-	4	5
TO-	-	-	-	-	2	3
RCR+	-	-	2	8	4	5
RCR-	-	-	2	6	4	5
RCA	-	-	2	3	4	5
RCN	-	-	2	3	4	5
RTR	-	-	2	3	4	5
RTA	-	-	2	3	2	3
RUC	-	-	2	3	4	5
RXJ+	-	-	2	3	4	5
RXJ-	-	-	2	3	2	3
RXR	-	-	2	3	4	5

FIG. 12B

1204

Events	State 6 Req-Sent	7 Ack-Rcvd	8 Ack-Sent	9 Opened
Up	-	-	-	-
Down	1	1	1	1
Open	6	7	8	tc3,9r
Close	4	4	4	4
TO+	6	6	8	-
TO-	3p	3p	3p	-
RCR+	8	9	8	8
RCR-	6	7	6	6
RCA	7	6	9	6
RCN	6	6	8	6
RTR	6	6	6	5
RTA	6	6	8	6
RUC	6	7	8	9
RXJ+	6	6	8	9
RXJ-	3	3	3	5
RXR	6	7	8	9

[p]	Passive option
[r]	Restart option
[s]	Silent option

// Transition conditions

tc1 - (enabledSilent() == TRUE)
tc2 - (enabledSilent() == FALSE)
tc3 - (enabledRestart() == TRUE)

FIG. 13

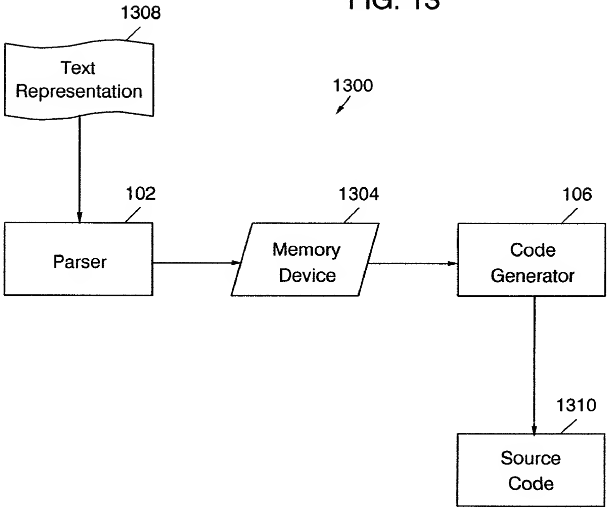

1308
Text Representation
1300
102
Parser
1304
Memory Device
106
Code Generator
1310
Source Code

FIG. 14

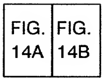


FIG. 14A

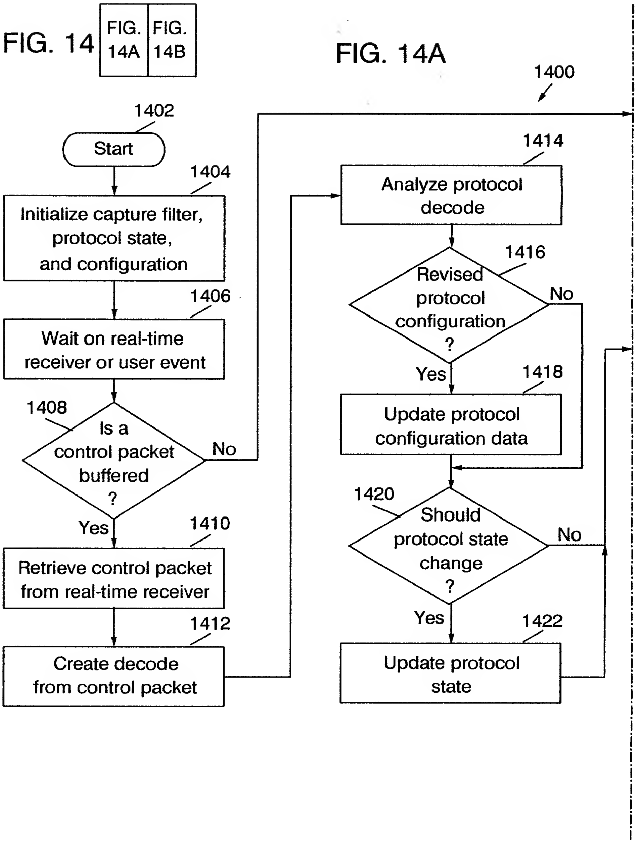

1400
1402
Start
1404
Initialize capture filter, protocol state, and configuration
1406
Wait on real-time receiver or user event
1408
Is a control packet buffered ?
No
Yes
1410
Retrieve control packet from real-time receiver
1412
Create decode from control packet
1414
Analyze protocol decode
1416
Revised protocol configuration ?
No
Yes
1418
Update protocol configuration data
1420
Should protocol state change ?
No
Yes
1422
Update protocol state

FIG. 14B

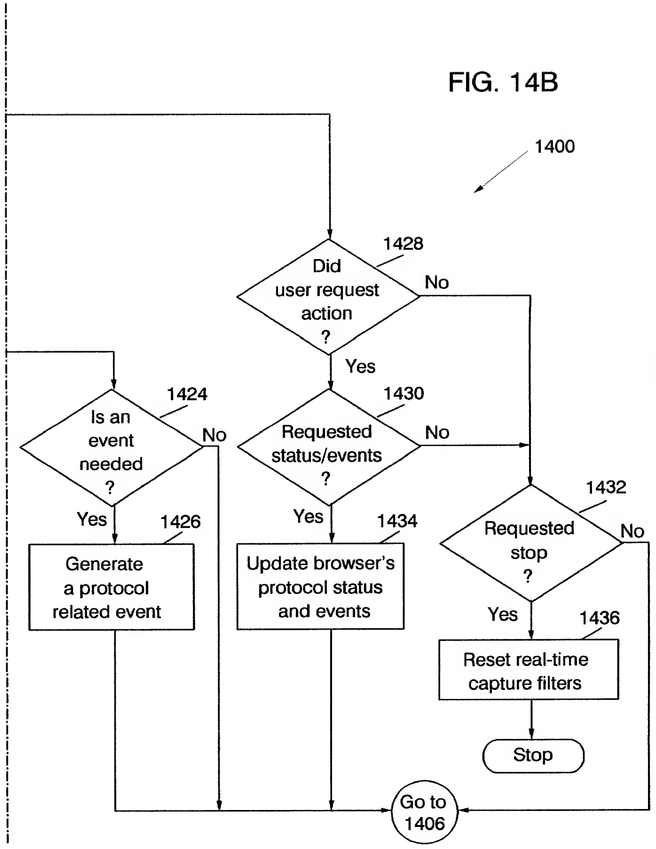

1400
1428
Did user request action ?
No
Yes
1430
Requested status/events ?
No
Yes
1424
Is an event needed ?
No
Yes
1426
Generate a protocol related event
1434
Update browser's protocol status and events
1432
Requested stop ?
No
Yes
1436
Reset real-time capture filters
Stop
Go to 1406

FIG. 15

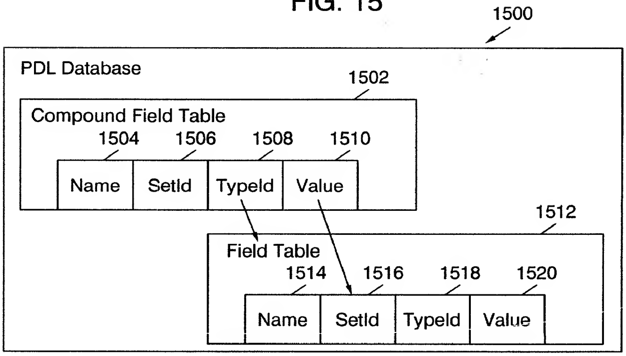
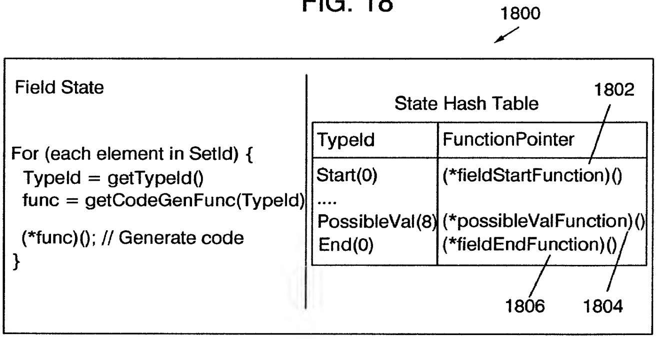


FIG. 18



```
For (each element in SetId) {
  TypeId = getTypeId()
  func = getCodeGenFunc(TypeId)

  (*func)(); // Generate code
}
```

TypeId	FunctionPointer
Start(0)	(*fieldStartFunction)()
....	
PossibleVal(8)	(*possibleValFunction)()
End(0)	(*fieldEndFunction)()

FIG. 16

1600

1602 1604 1606 1608

1610

TypeId	TypeName	TableName	Type	Comment
0	Start		Control	
0	ProtocolNames	ProtocolNames		
1	Protocol	Protocol	Compound	
2	Header	Header	Compound	
3	Payload	Payload	Compound	
4	Trailer	Trailer	Compound	
5	CompountField	CompountField	Compound	
6	Repeat	Repeat	Compound	
7	Switch	Switch	Compound	
8	PossibleValues	PossibleValues	Attribute	
9	Field	Field	Simple	
10	Len	Len	Attribute	
11	MinLen	Len	Attribute	
12	MaxLen	Len	Attribute	
13	Display	Display	Attribute	
14	Encode	Encode	Attribute	
15	Default	Default	Attribute	
16	Break	Len	Attribute	
17	Optional	Len	Attribute	
18	Offset	Len	Attribute	
19	Name	Name	Attribute	
20	Description	Description	Attribute	
21	String	String		
22	End	End	Control	
23	DecisiveField	Field	Simple	
24	FieldType	Attribute	Attribute	
28	MinVal	Attribute	Attribute	
29	MaxVal	Attribute	Attribute	
30	Count	Len	Attribute	

1612

FIG. 17

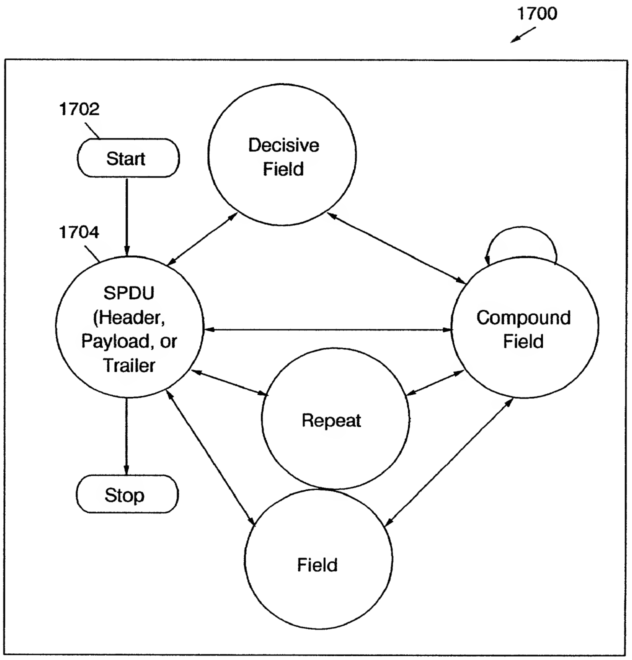
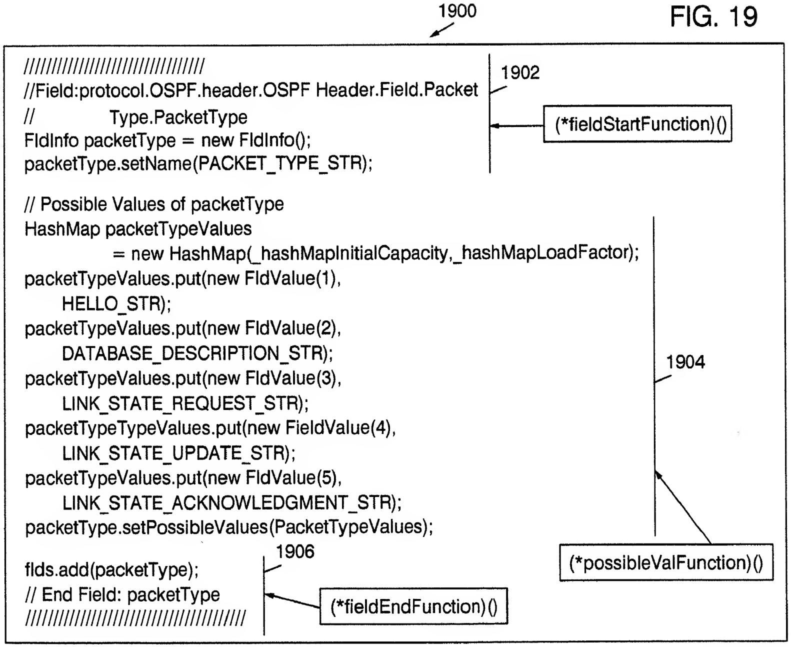

1700
1702
Start
Decisive Field
1704
SPDU (Header, Payload, or Trailer
Compound Field
Repeat
Stop
Field

FIG. 19

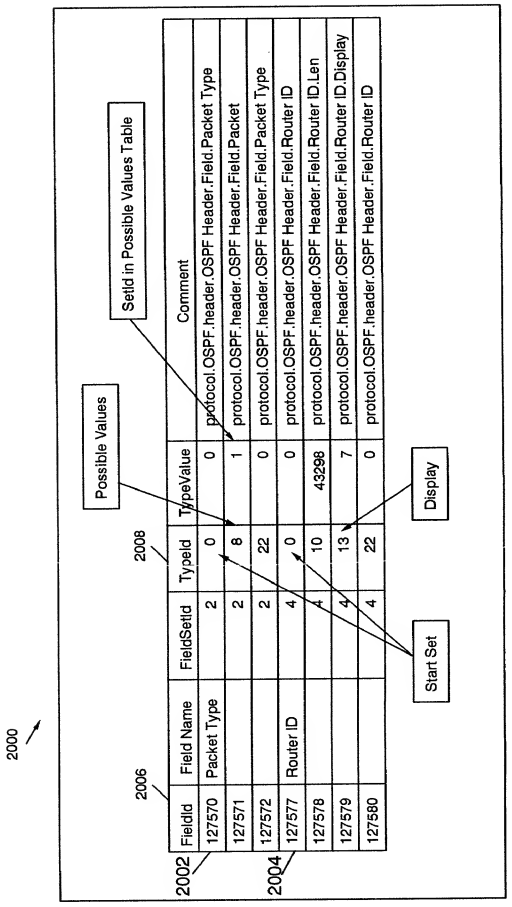


```
/////////////////////////////////
//Field:protocol.OSPF.header.OSPF Header.Field.Packet
//            Type.PacketType
FldInfo packetType = new FldInfo();
packetType.setName(PACKET_TYPE_STR);

// Possible Values of packetType
HashMap packetTypeValues
              = new HashMap(_hashMapInitialCapacity,_hashMapLoadFactor);
packetTypeValues.put(new FldValue(1),
      HELLO_STR);
packetTypeValues.put(new FldValue(2),
      DATABASE_DESCRIPTION_STR);
packetTypeValues.put(new FldValue(3),
      LINK_STATE_REQUEST_STR);
packetTypeTypeValues.put(new FieldValue(4),
      LINK_STATE_UPDATE_STR);
packetTypeValues.put(new FldValue(5),
      LINK_STATE_ACKNOWLEDGMENT_STR);
packetType.setPossibleValues(PacketTypeValues);

flds.add(packetType);
// End Field: packetType
/////////////////////////////////////
```

FIG. 20


2000
2002
2004
2006
2008
Possible Values
SetId in Possible Values Table
Display
Start Set

FieldId	Field Name	FieldSetId	TypeId	TypeValue	Comment
127570	Packet Type	2	0	0	protocol.OSPF.header.OSPF Header.Field.Packet Type
127571		2	8	1	protocol.OSPF.header.OSPF Header.Field.Packet
127572		2	22	0	protocol.OSPF.header.OSPF Header.Field.Packet Type
127577	Router ID	4	0	0	protocol.OSPF.header.OSPF Header.Field.Router ID
127578		4	10	43298	protocol.OSPF.header.OSPF Header.Field.Router ID.Len
127579		4	13	7	protocol.OSPF.header.OSPF Header.Field.Router ID.Display
127580		4	22	0	protocol.OSPF.header.OSPF Header.Field.Router ID

FIG. 21

Protocol	Status	Time	Mode
LCP	Open	09/04/00 08:01:03 AM	Emulate
IPCP	Negotiating	09/04/00 08:01:07 AM	Monitor
MPLSCP	Closed	09/04/00 08:01:05 AM	Monitor
RSVP	N/a	09/04/00 08:01:00 AM	Disabled

FIG. 22

	Rx1	Rx2
Current Status	Open	Negotiating
Loop-back	No	No
Unanswered Echo Requests	0	0
Maximum Receive Unit	512	1500
Asynchronous Character Map	0	0
Authentication Protocol	Unknown	Unknown
Quality Protocol	N/a	N/a
Protocol Field Compression	Off	Off
Address/Control Field Compression	Off	Off
Magic Number	0xFF	0x1FF
FCS Alternative	CCITT 32-bit	CCITT 32-bit

FIG. 23

FIG. 23A
FIG. 23B

FIG. 23A

Time	Recvr	Protocol	MsgType	Event	Synopsis
09/04/00 08:01:01 AM	Rx1	LCP	ConfigReq	Protocol Negotiating	ACComp:On,Pcomp:On,Magic.0x1ab82049
09/04/00 08:01:01 AM	Rx2	LCP	ConfigAck	Open Protocol	ACComp:On,Pcomp:On,Magic.0x4e3d9123
09/04/00 08:01:02 AM	Rx2	LCP	ConfigReq	Protocol Negotiating	ACComp:On,Pcomp:On,Magic.0x1ab82049
09/04/00 08:01:03 AM	Rx1	LCP	ConfigAck	Open Protocol	ACComp:On,Pcomp:On,Magic.0x1ab82049
09/04/00 08:01:04 AM	Rx2	IPCP	ConfigReq	Protocol Negotiating	Local IP: 198.85.38.199
09/04/00 08:01:06 AM	Rx1	IPCP	ConfigAck	Open Protocol	Local IP: 198.85.38.199
09/04/00 08:01:06 AM	Rx1	IPCP	ConfigReq	Protocol Negotiating	Local IP: 198.85.34.35
09/04/00 08:01:06 AM	Rx2	IPCP	ConfigAck	Open Protocol	Local IP: 198.85.34.35
09/04/00 08:01:10 AM	Rx2	MPLSCP	ConfigReq	Protocol Negotiating	
09/04/00 08:01:12 AM	Rx2	MPLSCP	TermReq	Close Protocol	
09/04/00 08:11:01 AM	Rx1	RSVP	Rx1	Rx1	Resv Request <session: 198.85.34.45 UDP port 14>

09/04/00 08:11:03 AM	Rx1	RSVP	Rx1	Rx1	Resv Confirm <session: 198.85.34.45 UDP port 14>
09/04/00 08:11:04 AM	Rx2	RSVP	Rx2	Rx2	Path Request <session: 198.85.38.199 UDD port 0x82A>
09/04/00 08:11:06 AM	Rx1	RSVP	Rx1	Rx1	Resv Error <session: 198.85.38.199 UDP port 0x82A>
09/04/00 09:21:10 AM	Rx2	RSVP	Rx2	Rx2	Path Request <session: 198.85.38.199 UDP port 0x82A>
09/04/00 09:21:12 AM	Rx2	RSVP	Rx2	Rx2	Resv Confirm <session: 198.85.38.199 UPD port 0x82A>
09/04/00 09:21:30 AM	Rx1	RSVP	Rx1	Rx1	Path Tear <session: 198.85.34.45 UPD port 14>
09/04/00 09:21:32 AM	Rx2	RSVP	Rx2	Rx2	Resv Tear <session: 198.85.34.45 UPD port 14>
09/04/00 09:21:32 AM	Rx2	RSVP	Rx2	Rx2	Resv Tear <session: 198.85.34.45 UPD port 14>
09/04/00 11:44:30 PM	Rx1	IPCP	TermReq	Close Protocol	
09/04/00 11:44:31 PM	Rx1	IPCP	TermAck	Close Protocol	
09/04/00 11::44:32 PM	Rx1	LCP	TermReq	Close Protocol	
09/04/00 11:44:33 PM	Rx2	LCP	TermAck	Close Protocol	

FIG. 23B